Robot Formations for Area Coverage

Jürgen Leitner
juxi.leitner@gmail.com

[1] Helsinki University of Technology, Finland
[2] Luleå University of Technology, Sweden
[3] The University of Tokyo, Japan

Abstract. Two algorithms for area coverage (for use in space applications) were evaluated using a simulator and then tested on a multi-robot society consisting of LEGO Mindstorms robots. The two algorithms are (i) a vector force based implementation and (ii) a machine learning approach. The second is based on an organizational-learning oriented classifier system (OCS) introduced by Takadama in 1998.

1 Introduction

This paper aims to describe the usage of a machine learning (ML) and evolutionary computing approach to use robot formations for area coverage, e.g. in wireless coverage and astronaut support. ML was chosen because of its capabilities with dynamic and unknown environments, the results will be compared with results based on a simple vector approach.

An interesting scenario for multi-robot cooperation in space is the exploration of the Moon and Mars, where rovers are currently on the forefront of space exploration. For permanent human settlement will depend heavily on robotic reconnaissance, construction and operational support. Tasks for the robots will include mapping landing sites, constructing habitats and power plants, communicating with and acting as a communication relay to Earth, and so forth. Scenarios for this assume many different robots, with different, but overlapping, capabilities working together. One scenario for the use of multiple robots teaming up and working together can be taken from the recent NASA plans for building *human outposts* on the Moon and Mars. These plans outline also a need for robotic support for the astronauts and explorers. In this scenario robots will, for example, search cooperatively for a location suitable in size and other properties to harbor a permanent human settlement. These tasks will include soil preparation and movement as well as, for example, carrying solar panels in tight-cooperation between two robots. The heterogeneity of the rovers is exploited throughout the whole mission to allow for better performance. Meanwhile, other rovers will begin with the exploration and surveying of the region around the construction site. Rovers with specialized sensing instruments are sent to investigate and cover as much of the area as possible, possibly transported in a larger mother-ship type robot at first. Formations of rovers will generate a wireless communication and emergency network for the robots as well as future human explorers.

Coverage Problem The *coverage problem* (sometimes also referred to as *covering problem*) is widely defined as to *"cover a search space consistently and uniformly"* (1), or more informally (in robotic exploration) to "see" every point of a given area. The coverage problem for a team of robots was categorized into three types (2): blanket coverage, barrier coverage, and sweeping coverage. Blanket coverage, which is the closest problem to the area coverage presented here, has the objective of maximizing the total covered area by a static arrangement.

Machine Learning Machine learning (ML) is using artificial intelligence algorithms to allow a system to improve over time using input data. ML is (i) useful when human expertise is not available, (ii) needed when solutions change over time or need to be adapted and (iii) helping to understand how humans learn.

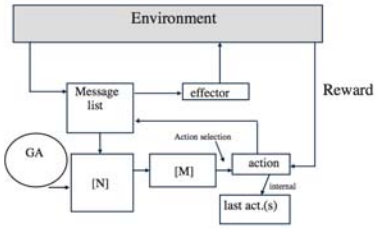
Evolutionary computing describes search algorithms based on natural selection and genetics. The idea is to search a problem space by evolving from a random starting solutions to better/fitter solutions, generated by selection, mutation and recombination.

This paper describes two implemented algorithms: a lightweight and simple force vector approach and a more complex algorithm based ML principles. They were implemented in C++ and tested in a separately developed simulator called *SMRTCTRL* (3).

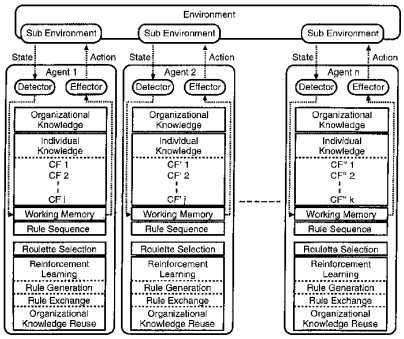
2 Related Work

The Art Gallery Problem is the real-world problem of guarding a gallery with a minimum number of guards, viewing the whole gallery. The mathematical problem was proved to be NP-hard (4).

The Learning Classifier System (LCS) developed by Holland in the 1970s is based on techniques found in evolutionary computing and reinforcement learning. It tries to learn which actions to perform based on the input received. LCS



(a) Holland's LCS (5)



(b) Takadama's OCS (6)

uses a population of binary rules on which a genetic algorithm (GA) is used to select and alter these (7). The GA evaluates rules by its fitness, which is based on a payoff received similar to reinforcement learning approaches.

An LCS-based algorithm receives *inputs* from the environment at every iteration step. Based on this the algorithm selects the best/fittest action. The possible *actions* depend on the (physical) context of the system. The *reinforcement* should reflect physical properties, e.g. a mobile robot might get reinforcement according to the distance moved towards the goal. An LCS is a rule-based system, with the *rules* usually in the form of "`IF state THEN action`". The GA used is operating on the whole rule-set every few time steps using roulette wheel selection and replacing rules with newly created ones.

Organizational-learning Oriented Classifier System (OCS) takes an *organizational learning* approach and adapts it to machine learning in multi-agent systems. The method is described as *"learning that includes four kinds of reinterpreted loop learning"* (6) and tries to generate a *hybrid solution*, aiming to overcome restrictions of the Michigan and Pittsburgh approaches (8). The system, sketched in Fig. 1b, uses reinforcements and evolutionary learning.

Vector-Based Formations The use of vectors is often found in reactive behaviors for robots, with this representation generally known as a *vector-force-field*. These approaches, generally, do not need a model of the environment, it is enough to sense obstacles. (9) presents an approach to use vector-force-fields to control the movement of robots in a given formation, while avoiding obstacles and progressing towards a target. It concentrates on the subtask of controlling and maintaining the formation by using forces calculated from the other robots, leading to the robot being pulled into the formation or pushed away. The paper uses position and heading of each robot to calculate 2 independent forces.

3 Placement Using Force Vectors

A vector-based approach, with vectors representing attractive and repulsive forces, is used to place the robots close to the target area. These forces are calculated and a combination of these is used to generate the movement for each robot separately. First the attractive forces to the mid-point of the target area and to the other robots are calculated, then repulsive forces are added, where the values are given by the coverage in the direction of the other robot. From this the coverage optimization is an emergent property of this algorithm.

4 Placement Using Machine Learning

ML techniques, based on the work of Takadama on OCS (6), are used to calculate the actions to be taken to optimally place the robots for area coverage. The system consists of autonomous *agents*, with local LCS-based implementations.

Each agent is able to recognize the *environment* and its local state and is able to change the environment due to a chosen action. The algorithm is implemented in a class encapsulating the learning algorithm and facilitates an object-orientated approach for representation of the agents and the environment. Each agent has its local memory, which is used to create, store and update rules, also called *classifiers* (CFs). These rules are used to select the most suitable action for the current state. The agents apply reinforcement learning (RL) and GA techniques for rule management. For this they detect the environment state and decide on an action based on the state. Each agent updates its own rule-set.

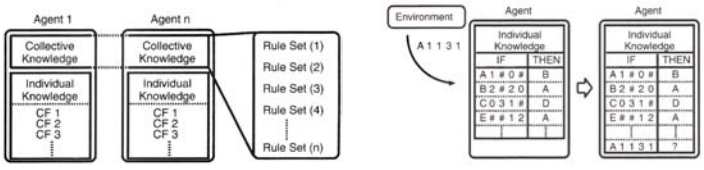
4.1 Mechanisms

The OCS algorithm uses 4 separate forms of *learning mechanisms* presented and described in this section. Pseudo-code is added to sketch the implemented algorithm and its main components as described in (6). The main OCS loop is shown in Listing 1.1.

Listing 1.1: The OCS algorithm

```
procedure OCS
    iteration := 0
    Collective_Knowledge_Reuse
    while iteration < MAX_ITERATION
        reset the problem to the starting position
        iteration := iteration + 1, step := 0
        while not solution_converges
            Detector()
            Rule_Generation, Rule_Exchange
            Effector()
            step := step + 1
        end
        Reinforcement_Learning, Collective_Knowledge_Reuse
    end
```

Collective Knowledge Reuse. The reuse of good solutions (e.g. role specialization) is helping to reduce the learning time as well as solving problems that



(a) Collective Knowledge Reuse (b) Rule Generation

Fig. 1: Illustration of 2 Learning Mechanisms in OCS (10).

need *organizational knowledge*. It uses a central set of agent behaviors, as is depicted in Fig. 1a. This mechanism (Listing 1.2) is useful when no solutions can be found without organizational knowledge, but due to time constraints was not implemented in the test runs.

Listing 1.2: `Collective_Knowledge_Reuse` procedure

```
procedure Collective_Knowledge_Reuse
    if iteration = 0
        use stored collective knowledge
    else if solution is the best
        if collective knowledge is set already
            delete stored collective knowledge
        store current rule-sets as collective knowledge
```

Rule Generation (Fig. 1b) is used to create *new* CFs when no CFs in the rule-set of an agent, i.e. the *Individual Knowledge Memory*, match the current sub-environment state detected. The generation of new CFs allows to explore other solutions to the problem, that have not yet been tested. These new CFs are created, with a random action value and a *condition* based on the state detected. The strength is initialized to `START_STRENGTH`. The weakest CF is dropped if the defined maximum of CFs is reached (Listing 1.3).

Listing 1.3: `Rule_Generation` procedure

```
procedure Rule_Generation
    for all agents
        if no classifier matched
            if number of classifiers = MAX_CF
                delete CF with lowest strength
            create a new CF & set strength to START_STRENGTH
    end
```

Rule Exchange is used to generate new behaviors in the agents and therefore explore new areas and CF combinations. The CFs of two, randomly chosen, i.e. non-elite selection, agents are exchanged, at every `CROSSOVER_STEP`. The rules are sorted by strength and the worst CFs of one agent are replaced by the best CFs of the counterpart and vice-versa (Listing 1.4). The strength values of the rules exchanged are then reset to the starting value. The rule exchange is depicted in Fig. 2a.

Reinforcement Learning (RL) (Fig. 2b) updates the *strength* of each classifier in the *Rule Sequence Memory* in the case of a converging result. The received reward is spread, based on a simple exponential function, over all the fired CFs, i.e. the CFs in the *Rule Sequence Memory*.

Listing 1.4: `Rule_Exchange` procedure

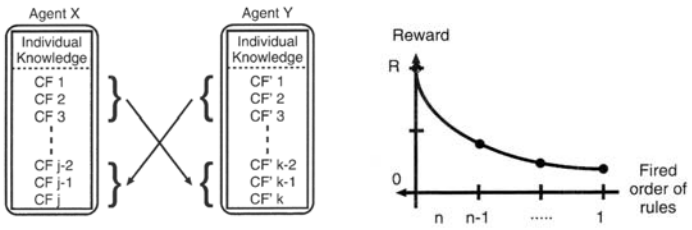
```
procedure Rule_Exchange
    if mod(step, CROSSOVER_STEP) = 0
        for a random pair of agents
            order rules by strength
            for the weakest CROSSOVER_NUM CFs
                if strength < BORDER_ST
                    delete CF
            copy CROSSOVER_NUM strongest from other agent
            reset strength to START_STRENGTH
```

4.2 Classifier

The classifiers are the same as the rules in LCS, simple *if-then* constructs composed of three parts: (i) the *condition*, (ii) the *action* and (iii) the *strength* or fitness value. The *condition*, or *if* clause, of the CF contains various information about the environmental state. The *action*, or *then* part, defines the action to be executed by the agent if the condition part matches the current environment. The third part, the *strength* value, defines how good the CF is, which effects selection and therefore the agent's behavior.

Condition The condition part of a classifier is representing the knowledge of the current situation. It is a list of input values, represented by the ternary alphabet $\{0, 1, \#\}$, though extended in some cases. Fig. 3 shows the structure of the CFs and the condition in the OCS algorithm. The condition contains: (i) The `PREV_ACTION` executed by this agent, (ii) a flag representing an existing `OVERLAP` with another agent, (iii) a flag representing a `LOC_CHANGE` and (iv) a number calculated and indicating the `PREV_CONDITION`.

Actions The actions of the agents are 7 simple motions, derived from a simple mobile robot. The actions are: MOVE FORWARD, MOVE BACKWARD, MOVE RIGHT, MOVE LEFT, ROTATE RIGHT, ROTATE LEF and STAY.



(a) Rule Exchange (b) Reward Spreading in RL

Fig. 2: Illustration of 2 Learning Mechanisms in OCS (10).

4.3 Memories

The memories at every agent store various information about the *classifiers*. In the memory CFs can be created, stored and updated. The four memories of each agent are:

(i) *Organizational Knowledge Memory*, also referred to as *Collective Knowledge Memory*, stores a global set of rule-sets. This knowledge is shared by all agents and allows for role specialization and classifier reuse.

(ii) *Individual Knowledge Memory* stores the current rule-set of every agent. This is where the knowledge of the individual agent is developed over the various iterations. `FIRST_CF` number of rules are generated and initialized at the start, then during the learning operation new rules are added or exchanged.

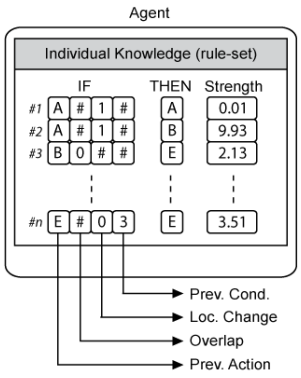
(iii) *Working Memory* is used for operations in the agents, e.g. selection of matching CFs, a temporary list is available to be filled with CFs.

(iv) *Rule Sequence Memory* stores the rules selected at every iteration.

4.4 Environment Interactions

The *environment interactions* are modeled and are implemented in two functions: (i) for retrieving the current state, `Detector()` and (ii) reacting by executing an action, `Effector()`.

Detector() This function detects the current state of the (sub-)environment at every agent, with the main objective to create a new CF condition representing the state. This is modeled into the 4-field vector, described above, where the `PREV_CONDITION` is calculated by adding the last iterations overlap field with the current. If one of them is set, also the value of the current `LOC_CHANGE` flag is added.



Listing (1.5): Code for the CFs

```
enum eCFCondElements {
  PREV_ACTION = 0, OVERLAP,
  LOC_CHANGE, PREV_CONDITION
};

//...
char cond[CDPARNR]={'#','#','#','#'};
// init others
cond[PREV_CONDITION] =
  prevOverlap - '0' + cond[OVERLAP];
if(cond[PREV_CONDITION] != '0' &&
   cond[LOC_CHANGE] == '1')
  cond[PREV_CONDITION]++;
prevOverlap = cond[OVERLAP];
```

Fig. 3: The structure of the rules, aka classifiers (CFs). *based on (6)*

5 Results

5.1 Simulation Results

A *standard initial configuration* (SIC) was used to be able to compare the results achieved in the test runs. This configuration reflects a marsupial society just deployed using: a *mother-ship* (Motherbot), which is, for these two control algorithms, not moveable and has a circular coverage area (5x5, 25 cells), as well as four controllable *child-bots* (Marsubots), each possessing an elliptic coverage area (5x7, 27 cells). The robots are placed with one cell in between each other. The *target area* is defined as a circular area, 69 (9x9) cells in size, and placed 22 and 11 cells away in X and Y axis respectively.

Vector-Based Simulation Results The vector-based control approach was mainly aimed to be tested with circular target areas and therefore does not implement a directional force vector, hence the robot actions are reduce to the four motions (*forward, backward, left, right*). For the simulation runs no obstacles were inserted and no terrain interaction was performed on the sensing areas. The simulation also assumed that all actions can be completed within one time step and that the robots can be controlled simultaneous, which turned out to be difficult (see SMURFS project below).

The vector approach generated higher repulsive forces, when switched to elliptical, due to the lack of a directional understanding, and hence not a full coverage was found. This tests, with the elliptic sensor areas, showed that the algorithm converges to a stable formation after 83 actions, generating a combined coverage of only 12%, or 8 of the 69 target area cells. This could be overcome by using a better scaling factor and a directional component.

An interesting effect observed was that robots with smaller sensing areas are moving closer to the target, and by doing so, drive the robots with larger areas away from it. This is explained by the target force calculation, which leads to larger forces in the direction of the target for robots with smaller coverage (since there is a smaller repulsive force).

Machine Learning All ML algorithm test runs use the SIC and classifier-action pairs selected through learning to control the robots' motion. After a convergence in the results, the robots are reset to the SIC and another iteration of ML is performed. Here all 7 robot motions were used.

Throughout the iterations the agents and therefore the global solutions are increasingly optimizing the coverage problem, with the the best coverage a bit over 26%, which is better than the vector-based approach. It can be seen that the results are varying quite a bit but an overall trend to increased coverage is visible. The results show that the randomized learning approach does overall yield a good coverage with a good distance and that the best solution improves over time.

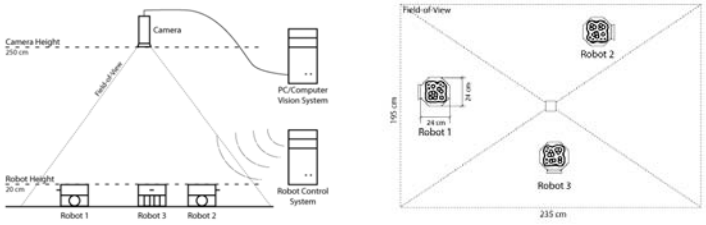
The test run used the following settings for its OCS learning to obtain the result: Each agent has between 15 (at start, `FIRST_CF`) and 30 (`MAX_CF`) classifiers and exchanges 7 at every crossover step (`CROSSOVER_CF_NUM`). These classifiers are initialized with a strength of 1.0 (`START_STRENGTH`) and the border strength (for crossover) is set to 1.15 (`BORDER_ST`). The test run had a maximum number for 500 iterations (`MAX_ITERATIONS`), each with a maximum of 300 steps (`MAX_STEPS`), but the average number of steps per iteration was only 68.3. A rule exchange between two random agents was done every 10 steps (`CROSSOVER_STEP`). The average reward per iteration was only 1.86 (with the maximum at 6.3).

As with the vector-based approach for the simulation no obstacles were inserted and the simulation assumed that all actions can be completed within one time step. A few hundred test runs were done, but due to the inherent randomness (in the initial rule generation) the results vary quite a bit, for example, the final coverage ranges between 0% and 47%. Obtaining the results was harder than anticipated and they are also not as good as expected.

5.2 Project *SMURFS* at IJCAI

To allow testing of the algorithm, robots from the Society of Multiple Robots (SMURFS) project were used, which were designed to be cheap and able to reconfigure. The robots were using LEGO Mindstorms NXTs for control and 4 motors for actuation. More information about the robots can be found at (11). A webcam and visual tracking system using fiducials (12), nicknamed *Gargamel*, was used to provide localization for the robots, which were then controlled via the *SMRTCTRL* simulator using Bluetooth. A simple sketch showing the experiment arrangement can be seen in Figure 4.

Experimental Findings The simple control, based on the vector-based placement algorithm, was controlling the robots using the freely available *nxtlibc* li-



(a) Camera Setup

(b) Field of View of the Camera, birds-eye view

Fig. 4: The *Gargamel* Tracking System for the *SMURFS* robots.

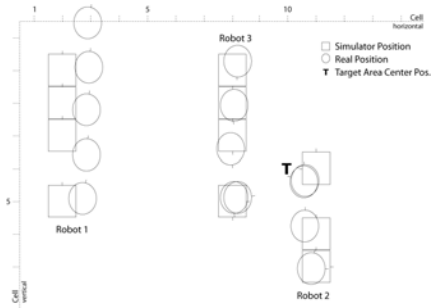


Fig. 5: Robot Movements: Real-World vs. Simulator

Table 1: A comparison of the results of the two algorithms.

	Vector Approach		**OCS**		
	(circular)	(elliptic)	(average)	(best)	(worst)
Coverage	100 %	12 %	26.58 %	46 %	0 %
Steps	29	83	64.03	17	289
Distance [px]	230	880	1235.3	582	1843

brary. The robots were controlled sequentially, due to limitations of the library, which did not provide multi-threading support. The discretization in the simulator was changed to closer represent the robots and the field-of-view of the camera, but even with that the robots could not be controlled very precisely. Though some effort was put into the action-programs, for example, the move forward program, it could not be ensured that the robot would move exactly one field when executed. This discrepancy was seen especially at the borders of the vision system. Videos of the system controlling the robots can be found online[4]. Figure 5 compares the motion of the robots as seen by the simulator *(boxes)* and in real-life *(circles)*. The data presented is the average of 3 test runs performed with the final revision. The position of the target is marked with a **T**. The graph shows the robots performing 12 actions (per test run), of which 3 are rotations. To visualize the orientation and rotation small lines are added to one side of the square or circle. Discretization errors at the center can be seen with all robots. The robots' `MOVE_FORWARD` motion were tested and yielded $20.1 - 20.5cm$, with the first movement usually a bit shorter ($19.2 - 19.6cm$), probably due to the initial orientation of the castor wheels.

In the first test runs a discrepancy between sent action and the outcome of the motion was seen. For example, a `MOVE_FORWARD` action would lead, due to drift, also to a movement to a side (this can also be seen in the motion of robot 1 in Figure 5); this was then reduced as much as possible.

[4] At our webpage (http://smrt.name/ijcai) and on YouTube (e.g. http://www.youtube.com/watch?v=RnP70SaH4iw). *(July 15, 2009)*

6 Conclusions

The *vector-based approach* leads to a converging coverage in most cases, though in some situations singularities or oscillations occur. It is not able to handle terrain interaction and performs best with circular sensor areas. The *organizational-learning oriented classifier system (OCS)* approach was more complicated to implement but allows for terrain interactions and different elevations as well as obstacles. It though is not always converging and is, in the simple case, outperformed by the lightweight vector approach (see Table 1). The system is still in its very early stages and needs to be evaluated further. Especially the connection between the parameters (for example, number of classifiers, strength values and of course, `MAX_ITERATIONS`, `MAX_STEPS`, `CROSSOVER_STEP`, just to name a few), but also the choice of the actions, the classifiers (and its structure) and the definition of the reward and fitness function need to be researched further. For real-world experiments the algorithm has to perform faster and as decentralized as possible, which is currently not the case.

To make the OCS approach work and yielding good results more time than planned was needed. In the end it worked better than the vector-based approach in an environment having different elevations and obstacles present. This is because of the classifiers, which allow the OCS approach to generate two solutions for a given input condition, for example, one that moves the robot to higher ground and a second one that keeps the robot on the same height. The fitness for the two solutions will not be the same, which means one of the CFs is chosen with higher probability, therefore the environment interaction leads to different solutions, which is not the case for the vector-based implementation. The conclusion would though still be that the lightweight, vector-based control algorithm might be the more feasible approach to use, although also this approach needs some further research to work in the environment and scenario described in this thesis.

At the IJCAI robotics workshop a simple vector-based control of the robots was shown. Due to time constraints only a very basic implementation was shown, but we are confident that with more time a more precise control using a multi-threaded Bluetooth library could be implemented.

7 Future Work

Apart from the placement problem discussed above, another interesting issue is the change of formation due to changing conditions. Some thoughts about possible extensions for the approaches to tackle this problem. Because of the limited time and increasing complexity this problem was though not solved in this project. The idea was to extend the ML approach to be able to handle these situations. This could be done by adding an algorithm, that works with the results of the OCS learner presented here and uses the optimal configurations found to generate a transition matrix. This matrix would contain which formation to switch to, according to some optimizing criteria, if one robot fails.

Bibliography

[1] Menezes, R., Martins, F., Vieira, F.E., Silva, R., Braga, M.: A model for terrain coverage inspired by ant's alarm pheromones. In: ACM Symp. on Applied computing. (2007) 728–732

[2] Gage, D.: Command control for many-robot systems. In: The 19th Annual AUVS Technical Symposium. (1992) 22–24

[3] Leitner, J.: Multi-robot formations for area coverage in space applications. Master's thesis, Helsinki University of Technology, Espoo, Finland (2009)

[4] Aggarwal, A.: The art gallery theorem: its variations, applications and algorithmic aspects. PhD thesis, The Johns Hopkins University (1984)

[5] Bull, L., Kovacs, T., eds.: Foundations of Learning Classifier Systems. Volume 183 of Studies in Fuzziness and Soft Computing. (2005)

[6] Takadama, K., Terano, T., Shimohara, K., Hori, K., Nakasuka, S.: Making organizational learning operational: Implications from learning classifier systems. Computational & Mathematical Organization Theory **5**(3) (1999) 229–252

[7] Holland, J.H.: Adaptation. In: Progress in Theoretical Biology. (1976)

[8] Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman (1989)

[9] Schneider, F.E., Wildermuth, D., Wolf, H.L.: Motion coordination in formations of multiple mobile robots using a potential field approach. In: Distributed Autonomous Robotic Systems 4. (2000) 305

[10] Takadama, K., Nakasuka, S., Shimohara, K.: Robustness in organizational-learning oriented classifier system. Soft Comput. **6**(3-4) (2002) 229–239

[11] Leal Martínez, D.: Reconfigurable multi-robot society based on lego mindstorms. Master's thesis, Helsinki University of Technology (2009)

[12] Kaltenbrunner, M., Bencina, R.: reactivision: A computer-vision framework for table-based tangible interaction (2007)